**深 圳 大 学 实 验 报 告**

**课程名称：­ 算法设计与分析**

**实验项目名称： 排序算法性能分析**

**学院： 计算机与软件学院**

**专业： 计算机科学与技术**

**指导教师： 马里佳**

**报告人： 钟善扬 学号：2017303031班级： 02**

**实验时间： 2020/04/08**

**实验报告提交时间： 2020/05/05**

**教务处制**

1. **实验目的**

* 掌握选择排序、冒泡排序、合并排序、快速排序、插入排序算法原理
* 掌握不同排序算法时间效率的经验分析方法，验证理论分析与经验分析的一致性。

1. **实验内容与过程**

2.1 问题描述

* 输入：无序数组
* 输出：有序数组

2.2 算法原理描述与核心伪代码

* 2.2.1 选择排序Selection Sort

2.2.1.1算法原理描述

* Step 1在未排序序列中找到最小（大）元素，存放到排序序列的起始位置。
* Step 2再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。
* Step 3重复Step 2，直到所有元素均排序完毕。

2.2.1.2核心伪代码

Selection\_Sort(A,N)

for i = 1 ~ N

Min\_i = i

for j = i ~ N

if A[j] < A[Min\_i]

Min\_i = j

swap(A[i], A[Min\_i])

* 2.2.2 冒泡排序 Bubble Sort

2.2.2.1算法原理描述

* Step 1比较相邻的元素。如果第一个比第二个大，就交换他们两个。从头到尾对每一对相邻元素作同样的工作。这步做完后，最后的元素会是最大的数。
* Step 2针对所有的元素重复以上的步骤，除了最后一个。
* Step 3持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

2.2.2.2核心伪代码

Bubble\_Sort(A,N)

for i = 1 ~ N

for j = 1 ~ N - i

if A[j] > A[j+1]

swap(A[j], A[j+1])

if no\_more\_exchange

Break

* 2.2.3 插入排序 Insertion Sort

2.2.2.1算法原理描述

* Step 1将第一待排序序列第一个元素看做一个有序序列，把第二个元素到最后一个元素当成是未排序序列。
* Step 2从头到尾依次扫描未排序序列，将扫描到的每个元素插入有序序列的适当位置。如果待插入的元素与有序序列中的某个元素相等，则将待插入元素插入到相等元素的后面。
* Step 3重复Step 2，直到未排序序列为空。

2.2.2.2核心伪代码

Insertion\_Sort(A,N)

for i = 2 ~ N

node = A[i]

for j = i - 1 ~ 1

if(A[j]> A[i])

A[j+1] = A[j]

else break

A[j+1] = node

* 2.2.4 合并排序 Merge Sort

2.2.2.1算法原理描述

* Step 1对二分的左右两个序列进行归并排序；
* Step 2申请空间，使其大小为两个已经排序序列之和，该空间用来存放合并后的序列；
* Step 3设定两个指针，最初位置分别为两个已经排序序列的起始位置；
* Step 4比较两个指针所指向的元素，选择相对小的元素放入到合并空间，并移动指针到下一位置；
* Step 5重复步骤 3 直到某一指针达到序列尾，即可将另一序列剩下的所有元素直接复制到合并序列尾。

2.2.2.2核心伪代码

Merge\_Sort(left, right)

if(left == right)

return

middle = (left + right) / 2

Merge\_Sort(left, middle)

Merge\_Sort(middle + 1, right)

Merge(left, right)

* 2.2.5 快速排序 Quick Sort

2.2.2.1算法原理描述

* Step 1从数组中选择一个基数node进行分割，一般选择头元素。将小于该基数的数组元素放在该基数左边，将大于该基数的元素放在该基数右边。
* Step 2将分割出来的两个子数组分别选择基数进行分割。
* Step 3直到分割出来的所有子数组只有一个元素则表示排序结束。

2.2.2.2核心伪代码

Quick\_Sort(A, left, right)

node = A[left]

i = left

j = right

while(i < j)

While(i < j && A[j] >= node)

j--

else

A[i] = A[j]

while(i < j && A[i] < node)

i++

else

A[j] = A[i]

if(left < i - 1)

Quick\_Sort(left, i - 1)

if(right > i + 1)

Quick\_Sort(i + 1, right)

2.3 算法测试结果及效率分析

* 2.3.1 选择排序结果分析

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数组长度 排序方法 | 5000 | 10000 | 15000 | 20000 | 25000 | 30000 | 35000 | 40000 | 45000 | 50000 |
| selection\_sort | 0.0312 | 0.119 | 0.2652 | 0.4612 | 0.733 | 1.0916 | 1.4646 | 1.8812 | 2.4394 | 3.0118 |

* 算法复杂度理论分析：

每次循环分别比较N - 1次，N-2次，N-3次，……，共比较的次数是 (N - 1) + (N - 2) + ... + 1。求和，得N（N-1）/ 2，其时间复杂度为 O(N^2)。

* 结果分析：

图中使用2阶多项式函数拟合，可决系数R^2高达0.9998，表明实测值和理论分析的变化趋势几乎相同。

* 2.3.2 冒泡排序结果分析

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数组长度 排序方法 | 5000 | 10000 | 15000 | 20000 | 25000 | 30000 | 35000 | 40000 | 45000 | 50000 |
| bubble\_sort | 0.0934 | 0.396 | 0.8966 | 1.6092 | 2.5366 | 3.7618 | 5.151 | 6.6584 | 8.4744 | 10.5122 |

* 算法复杂度理论分析：

若不考虑优化的情况，则外层循环执行 N - 1次，内层循环最多的时候执行N次，最少的时候执行1次，平均执行 (N+1)/2次，一共执行 (N - 1)(N + 1) / 2 = (N^2 - 1)/2次，故复杂度为O(N^2)。

* 结果分析：

图中使用2阶多项式函数拟合，可决系数R^2高达0.9999，表明实测值和理论分析的变化趋势几乎相同。

* 2.3.3 插入排序结果分析

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数组长度 排序方法 | 5000 | 10000 | 15000 | 20000 | 25000 | 30000 | 35000 | 40000 | 45000 | 50000 |
| insertion\_sort | 0.015 | 0.059 | 0.1318 | 0.2358 | 0.3676 | 0.5686 | 0.7506 | 0.9644 | 1.239 | 1.5374 |

* 算法复杂度理论分析：

最优的情况是当待排序数组是有序时，只需当前数跟前一个数比较一次即可，一共需要比较N-1次，时间复杂度为O(N)

最坏的情况是待排序数组是逆序的，此时需要比较总次数为1+2+3+…+N-1，所以，插入排序最坏情况下的时间复杂度为O(N^2)

平均来说，A[1..j-1]中的一半元素小于A[j]，一半元素大于A[j],故平均时间复杂度依然为O(N^2)

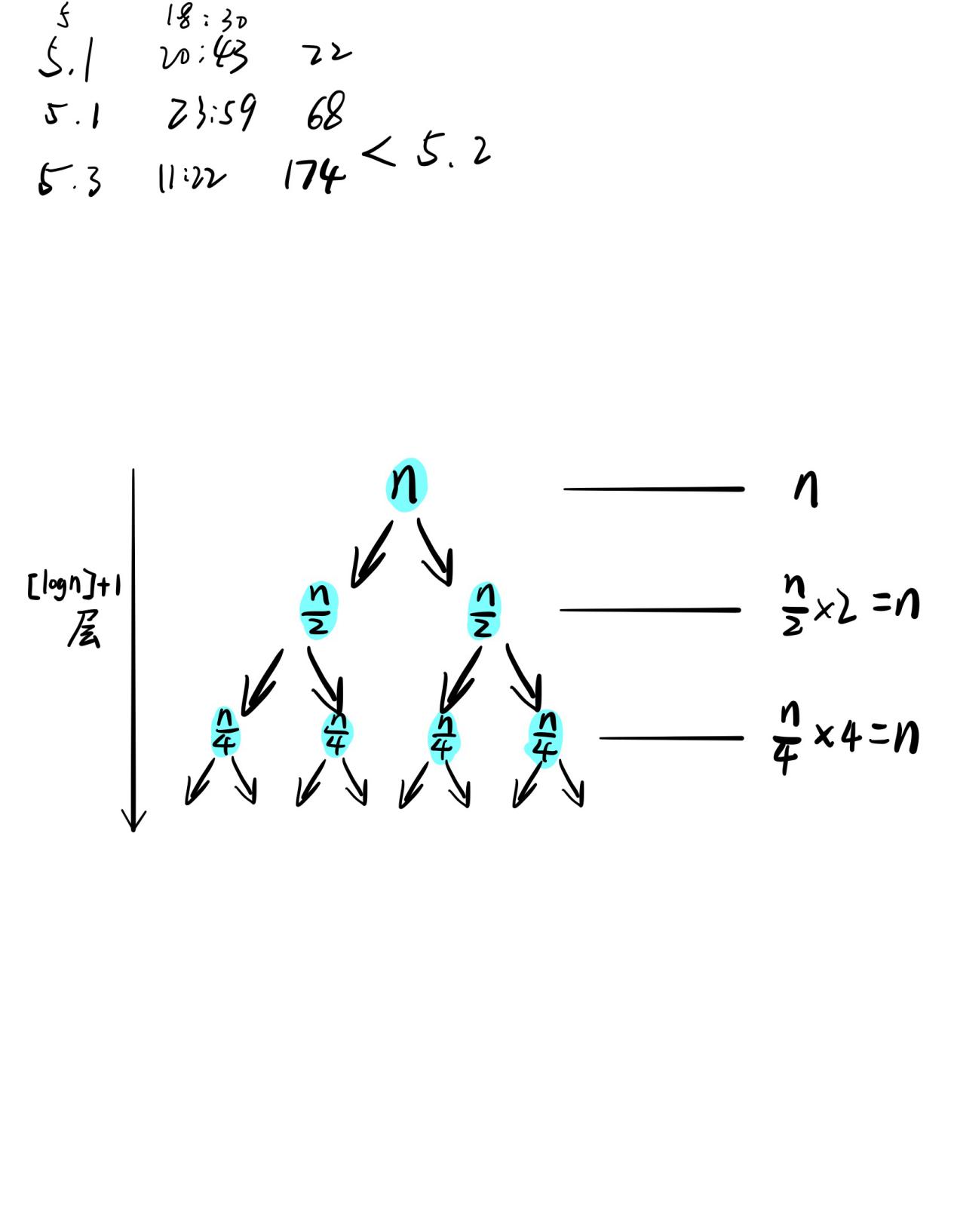
* 结果分析：

图中使用2阶多项式函数拟合，可决系数R^2高达0.9997，表明实测值和理论分析的变化趋势几乎相同。

* 2.3.4 合并排序结果分析

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数组长度  排序方法 | 5000 | 10000 | 15000 | 20000 | 25000 | 30000 | 35000 | 40000 | 45000 | 50000 |
| merge\_sort | 0.021 | 0.087 | 0.141 | 0.0672 | 0.2046 | 0.3004 | 0.3484 | 0.3712 | 0.4322 | 0.5034 |

* 算法复杂度理论分析：



从这个递归树可以看出，第一层时间代价为n，第二层时间代价为n/2+n/2=n，不难发现往下每层代价均为n。共有[logn]+1层,故总的时间代价为n\*(logn+1)，时间复杂度是O(nlogn)

* 结果分析：

从图中可以看出对于万级数据，归并排序可在毫秒级完成，而曲线存在波动，推测是由于相对于归并排序来说，万级数据规模太小，运行时间太短，当数据比较小时即使偏差很小表现出来的误差还是很大，所以对实验环境带来的扰动非常敏感。

为了进一步验证理论分析的正确性，将数据调至十万级，结果如下。使用nlog(10)n+c函数相比较。可见此时实测值与理论趋势基本吻合。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 100000 | 200000 | 300000 | 400000 | 500000 |
| nlog(10)n+c | 1.67260002 | 3.513017575 | 5.416715165 | 7.361670221 | 9.337289612 |
| merge\_sort | 1.6726 | 3.4598 | 5.6283 | 7.4715 | 9.7206 |

* 2.3.5 快速排序结果分析

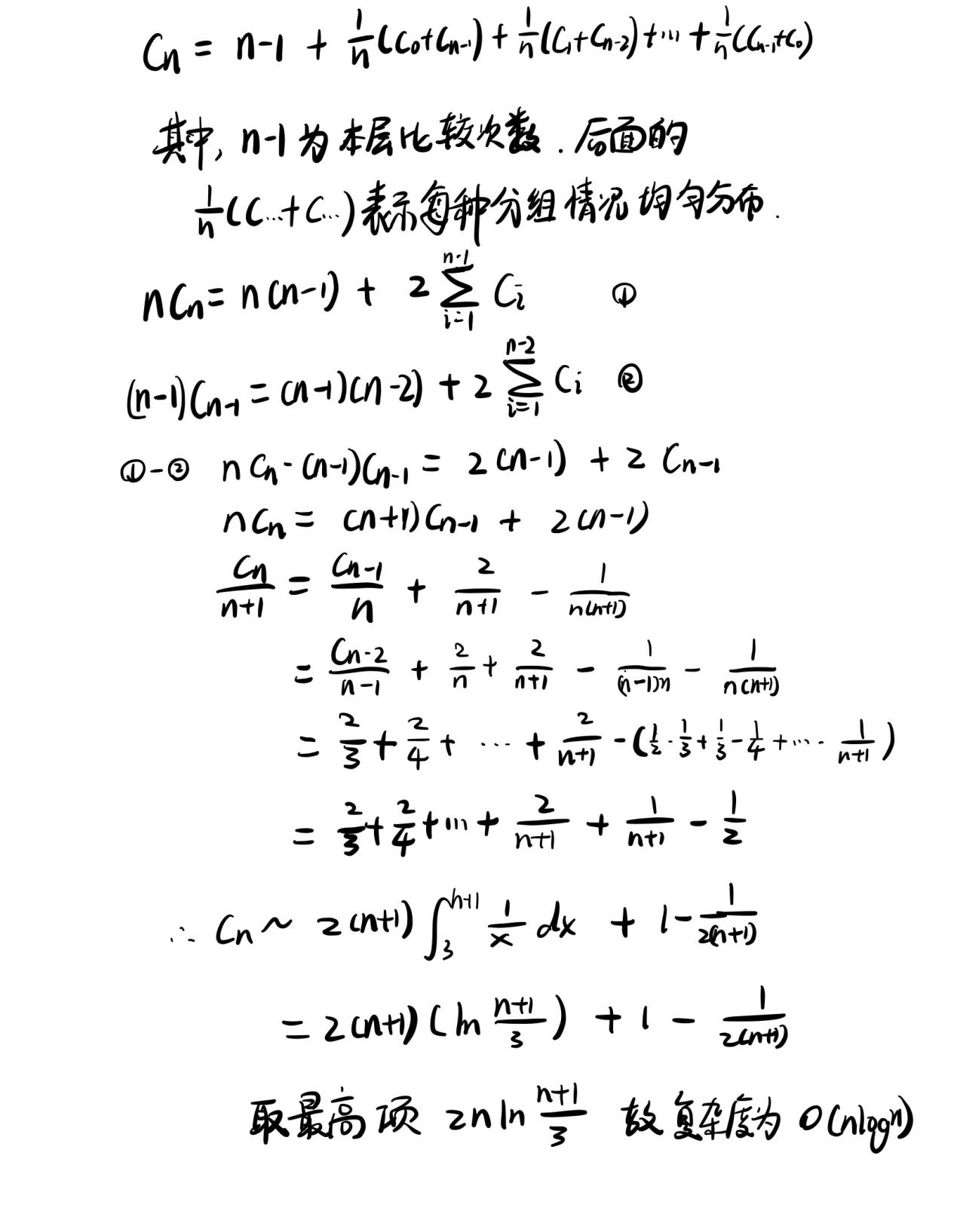
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数组长度 排序方法 | 5000 | 10000 | 15000 | 20000 | 25000 | 30000 | 35000 | 40000 | 45000 | 50000 |
| quick\_sort | 0 | 0.0004 | 0.0002 | 0.0002 | 0.0002 | 0.0006 | 0.0006 | 0.0006 | 0.0008 | 0.0006 |

* 算法复杂度理论分析：

在最优情况下，左子树和右边子树严格二分，递归树的深度为log（2）n + 1，即仅需递归log2n次，需要时间为T（n）。此时分析过程与归并排序相同，时间复杂度为O(nlogn)

在最坏的情况下，待排序的序列为正序或者逆序，递归树为长为n的直线。所以需要递归n‐1次，每次比较次数为N - 1次，N-2次，N-3次，……，共比较的次数是 (N - 1) + (N - 2) + ... + 1。求和，得N（N-1） / 2，其时间复杂度为 O(N^2)。

平均的情况，设平均比较Cn次，那么：



* 结果分析：

和归并排序情况一样，曲线存在波动，推测是由于数据规模太小，运行时间太短，当数据比较小时即使偏差很小表现出来的误差还是很大，对实验环境带来的扰动非常敏感。

同样将数据调至十万级，结果如下。使用nlog(10)n+c函数相比较。可见此时实测值与理论趋势基本吻合，但依然有波动，原因是快排本身对数据的起始排列顺序等因素很敏感，有时可能遇到递归树失衡的情况。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1000000 | 2000000 | 3000000 | 4000000 | 5000000 |
| nlog(10)n+c | 0.011300588 | 0.023735002 | 0.036596954 | 0.049737654 | 0.063085532 |
| quick\_sort | 0.0113 | 0.0259 | 0.04285 | 0.04725 | 0.06825 |

* 2.3.6 五种算法比较

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数组长度 排序方法 | 5000 | 10000 | 15000 | 20000 | 25000 | 30000 | 35000 | 40000 | 45000 | 50000 |
| selection\_sort | 0.0312 | 0.119 | 0.2652 | 0.4612 | 0.733 | 1.0916 | 1.4646 | 1.8812 | 2.4394 | 3.0118 |
| bubble\_sort | 0.0934 | 0.396 | 0.8966 | 1.6092 | 2.5366 | 3.7618 | 5.151 | 6.6584 | 8.4744 | 10.5122 |
| insertion\_sort | 0.015 | 0.059 | 0.1318 | 0.2358 | 0.3676 | 0.5686 | 0.7506 | 0.9644 | 1.239 | 1.5374 |
| merge\_sort | 0.021 | 0.087 | 0.141 | 0.0672 | 0.2046 | 0.3004 | 0.3484 | 0.3712 | 0.4322 | 0.5034 |
| quick\_sort | 0 | 0.0004 | 0.0002 | 0.0002 | 0.0002 | 0.0006 | 0.0006 | 0.0006 | 0.0008 | 0.0006 |

在数据规模较大时归并排序和快速排序两种时间复杂度较低的算法明显优于其他算法。

三种O(n^2)的算法比较：冒泡排序交换操作最多，因此耗时最多，时间复杂度的二次项系数也比其他两种更大。选择排序不急于调换位置，记录最小者，每次只进行一次交换，所以耗时比冒泡排序小。插入排序比选择排序花费的时间大约少了一半，这是因为选择排序需要全部比较，插入排序可能会有最优或者较优情况出现。

2.4 海量数据排序

通过前面的分析，可知归并排序和快速排序的速度在数据规模很大时由于算法复杂度更低，所以耗时明显要比其他排序算法少。而两者中快排又更胜一筹，因为快排的内存写操作更少，我们平时考虑时间复杂度的时候并不考虑这些常量时间的影响，但有时候常量的影响不可忽略。

所以考虑使用快速排序，但结果耗时长达1212742毫秒，即约1213秒：



而归并排序却仅需24秒：



经检查，发现引起该异常的原因是使用rand()函数生成的整数最大值不超过40000，由鸽笼原理可知有大量重复数据，这使得快排的递归树极不平衡，接近最坏情况。

但现实中，海量数据的取值范围是一个小范围整数的情况很普遍，比如各种考试排名，某省高考考生数量可达十万级，但分值却是在0~750之间取，而且服从高斯分布，每一分的可能分布有几百人；这时使用快排并不是最好的。

针对这种情况，我对快排进行了改进，引入策略改变阈值checker，每次递归如果发现有node位置在头或者尾，而且当前序列还比较长(大于a)时，说明快排此时不能高效分割，将出现单侧子树，checker递增1，当checker大于b且当前序列长度小于c时，改用插入排序。

之所以选择改用插入排序，是因为当前序列很可能已经有序或接近有序，这时可以接近插入排序的最优情况O(N)。

核心伪代码：

if ((node\_index == left || node\_index == right)

&& (right - left > a))

checker++;

bool change\_condition = checker > b && right - left < c;

...

if (change\_condition)

insertion\_sort(data, left, node\_index - 1);

insertion\_sort(data, node\_index + 1, right);

else

quick\_sort(data, left, node\_index - 1);

quick\_sort(data, node\_index + 1, right);

该方法运行耗时约为13秒：



使用系统sort函数耗时约为22秒：



这验证了该方法的有效性。

但该方法的不足之处在于，参数a,b,c是我多次尝试得到的，1亿数据的情况下取值为（1000, 300, 5000）时效果较好。由于我找了很久并未找到三个参数的最优设置关于输入规模N的函数，虽然尝试了几种不同的函数关系，但效果都不理想，因此无法进行自动调整。

一种可能的解决办法是通过取不同输入规模，然后尝试出一系列最优（a, b, c）——N对，然后画图拟合曲线。由于时间关系，以后有时间我将继续研究。

1. **经验总结**

1. 数据规模足够大时，效率从高到低：

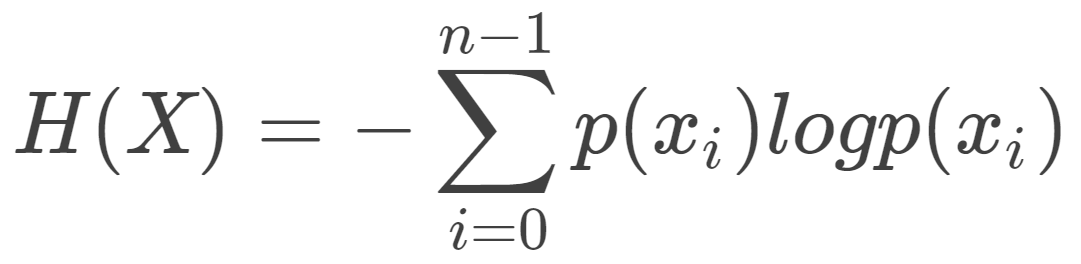
（归并排序，快速排序）> 插入排序 > 选择排序 > 冒泡排序

1. 数据大量重复，或者大量有序时，快排效率较归并排序差；否则快排效率比归并排序高
2. 为避免数据大量重复或者大量有序时快排效率较差的问题，可引入checker变量检测单侧子树出现次数，在一定条件下改用插入排序。
3. 除了使用C++实现了五种算法测试，我也使用Python进行了测试，结果发现Python比C++慢了几百倍。说明C++更接近机器语言。
4. **其他思考**

* **排序过程能不能比O（nlogn）更低？**

最近我接触到一门很厉害的学科——信息论

一个离散型随机变量X的熵H(X)的计算公式如下：



其中X0,X1,...,Xn-1为X的所有可能取值，p(x)是X事件发生的概率。如果log数以2为底，信息熵的单位为比特（bit）

排序问题中，N个数据的先后顺序是随机的，可以假设所有N！种组合情况概率相等。信息熵则为H(x)=log(n!)，而根据斯特林公式，log(n!)≈nln(n)-n,所有O(log(n!)）= O（nlogn）（周毅敏，李光耀《一种根据决策树结合信息论的经典算法复杂度可能下界分析》计算机科学 2013年11月 第40卷 第11A期）即对于基于比较的排序算法而言，假设我们一次比较所得到的平均信息量是1bit，那么要想消除信息的不确定性至少需要获得（nlogn）bit数据，也就是至少要比较nlogn次。

有了这种分析方法，我们对问题的求解和优化就有了强大的理论指导。比如，回到课程伊始，类似“在1000桶酒里选出有毒的那一桶酒”这种问题，我们知道每桶酒有毒概率相等，直接代公式H(X)=log2（1000）≈ 9.96，可知为了消除不确定性需要9.96bit的数据，所以需要至少10只老鼠，我们就不用再想能不能用少于10只的老鼠解决了。

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  年 月 日 |
| 备注： |

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。